Test Aspects of the JPL Viterbi Decoder

M. A. Breuer[1]
University of Southern California, Department of Electrical Engineering

This article deals with the generation of test vectors and design-for-test aspects of the JPL VLSI Viterbi decoder chip. Each processor IC contains over 20,000 gates. To achieve a high degree of testability, a scan architecture is employed. The logic has been partitioned so that very few test vectors are required to test the entire chip. In addition, since several blocks of logic are replicated numerous times on this chip, test vectors need only be generated for each block, rather than for the entire circuit. These unique blocks of logic have been identified and test sets generated for them. The approach employed for testing was to use pseudo-exhaustive test vectors whenever feasible. That is, each cone of logic is tested exhaustively. Using this approach, no detailed logic design or fault model is required. All faults which modify the function of a block of combinational logic are detected, such as all irredundant single and multiple stuck-at faults.

I. Introduction

The Jet Propulsion Laboratory (JPL) is currently designing a new Long Constraint Length VLSI Viterbi Decoder to be used on many future NASA missions [1]. This decoder consists of 8,192 Viterbi butterfly processors. A Viterbi decoder processor IC contains 16 Viterbi butterfly processors, resulting in over 20,000 gates per chip, with each individual butterfly processor having a complexity of about 1,800 gates. To enhance testability, a scan architecture [2] has been used. In this article we first discuss how the processor can be subdivided into three major blocks. Then the test architecture of each block is discussed along with the resulting test vectors required to test each block. Modifications to the logic which will simplify testing are also mentioned.

[1]The author is a consultant to JPL's Communications Systems Research Section.

II. Architecture

Figure 1 shows the hierarchical design schema of the Viterbi decoder chip. Entities in ovals represent macros. Entities in rectangles represent units of logic to be tested, such as gates, flip flops, multiplexers, or full adders. A number in brackets, such as $[n]$, indicates that there are n such entities. For example, a VC (Viterbi chip) macro consists of one 16-BFLYS macro and two MI macros. Table 1 indicates the gate-flip/flop (F/F) complexity of the main logic blocks in this chip. Blocks A–H are identified in Fig. 1. Assuming a flip flop consists of about 10 gate equivalences, this chip consists of approximately 20,000 gates.

The test generation for the Viterbi chip is based upon the analysis of three major blocks and related logic, namely the Metric Computer, the Memory Interface, and the Add-Compare-Select units. Each will be discussed in a separate section.

A *cloud* of logic is defined to be a combinational logic circuit all of whose outputs are either inputs to flip flops or are primary outputs, and all of whose inputs are either primary inputs or outputs of flip flops.

Note that a cloud of logic can be tested independently of any other combinational logic. Also, if a cloud of logic is replicated, then the tests for one cloud can be used to test all the other replicated clouds.

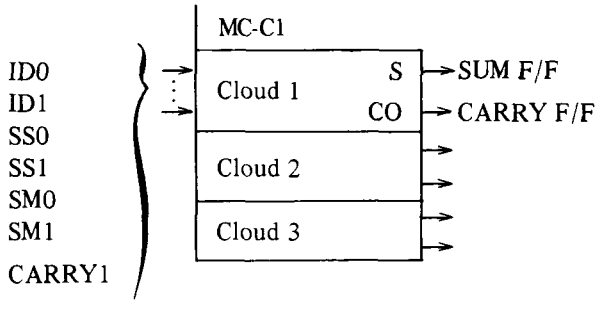
A *cone* of logic is defined to be a single-output combinational logic block whose output is either a primary output or an input to a flip flop, and whose inputs are either primary inputs or outputs of flip flops; every gate in the circuit which has a path through combinational logic to the output is in the cone.

A block of logic is said to be tested *pseudo-exhaustively* if an exhaustive test set is applied to each cone of the block.

III. Metric Computer

The architecture of the Metric Computer is shown in Figs. 2 and 3. Because of the feedback introduced by the carry flip flops, pipeline testing cannot be used. All flip flops are part of a scan chain. The combinational logic can be partitioned into four major blocks, namely MC-C1, MC-C2, MC-C3, and MC-C4 (see Fig. 3). The last three consist of only a full adder, and hence can be tested exhaustively with 8 test vectors. MC-C2 and MC-C3 have scan flip flops as drivers and receivers. MC-C4 has one primary input; the other I/O are scan flip flops.

MC-C1 has an architecture which can be decomposed into 3 clouds, as shown below.



MC-C1 has 7 X 3 = 21 inputs. To test MC-C1 exhaustively would require 2^{21} test vectors. However, each cloud has 7 inputs and can be tested exhaustively by $2^7 = 128$ test vectors. Due to the nature of the design, a pseudo-exhaustive test of just 8 test vectors exists. The test-vector set for the cloud shown in Fig. 4 is given in Table 2. The tests have been ordered so that C0 equals the next value of C, but this is not necessary. Note that when SM0 = 1, G1 is tested exhaustively; for SM1 = 1, G2 is tested exhaustively. Due to the fact that C0 implements a parity function, a sensitized path exists from G1 and G2 to a scan output. Thus, this test is a pseudo-exhaustive test for this cloud.

In summary, the Metric Computer can be tested with just 8 test vectors. The test is carried out as follows. A test vector is loaded into the scan flip flops. Simultaneously a test vector is loaded into the BFLY-ID shift registers. These two test vectors must be synchronized and aligned so that at time t, both scan chains are loaded. Then a normal clock is issued and all scan flip flops are loaded via their D input. The scan chain is then scanned out and the data checked. There are many ways for chaining flip flops to form a scan chain. The scan flip flops in the 16 Metric Computers can be put into one scan chain and then all Metric Computers can be tested as a unit by 8 test vectors. One Metric Computer has 6 flip flops in the BFLY-ID and 14 internal scan flip flops. The scan chain has 16 X 14 = 224 flip flops. Testing of the Metric Computers would take 8 + (224 X 8) = 1,800 clock cycles. The 8 comes from the 8 parallel-load clock cycles.

Figure 5 shows one possible scan path for this circuit.

Figure 6 indicates the BFLY-ID architecture. This circuit consists of one 6-bit shift register (SR) per Metric Computer. The 16 registers are connected together to form one long shift register. Only D flip flops are used; they are not scan flip flops and thus form what we refer to as a pseudo-scan chain.

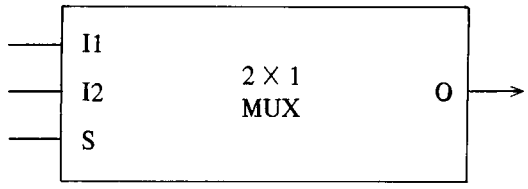
IV. Memory Interface Unit

There are two Memory Interface (MI) units. Each consists of a 16-bit parallel load shift register, as shown in Fig. 7(a). This unit both shifts and parallel loads as part of its normal operation, hence it has a unique $\overline{T}/R$ line, labeled LOAD. The register is made up of scan flip flops; the D inputs are used for parallel load; the scan-in for shifting data. This makes a double scan chain unnecessary. The four MIs share a common reset, clock, and load line. The parallel-in lines are driven by SELECT 0 ⟨31..16⟩; SELECT 0 ⟨15..0⟩; SELECT 1 ⟨31..16⟩; and SELECT 1 ⟨15..0⟩. The first SIN line to the unit should be tied to VDD or VSS. The RESET can be tested by loading in a vector consisting of all ones, resetting the flip flops, and scanning out the data and checking for all zeros.

The logic which drives each line is shown in Fig. 7(b) and consists of a scan flip flop and a MUX. There are 32 of these units. The architecture for the MI units and the logic which drives these units is shown in Fig. 7(c). The scan chain for the 12P flip flops is not identical to that shown in this figure. The

MUXs and MI units are tested by shifting a test vector into the register consisting of the 12P flip flops in the Compare-Select logic (CSL), passing it through the MUXs into the MI units, and shifting the data out of the MI units.

A 2 X 1 MUX is shown below.



A test for this device is shown below. S = 0 selects input I1; S = 1 selects input I2.

S	I1	I2	
0	0	1	Select I1 and pass a 0
0	1	0	Select I1 and pass a 1
1	0	1	Select I2 and pass a 0
1	1	0	Select I2 and pass a 1

To test the parallel load of a flip flop a 0 and a 1 are loaded. If the layout places lines close together in forming a register and there are possibilities of shorts, then an MI register can be loaded with the vectors 0101. . .01 and 1010. . .10. To test a shift register it is customary to pass a 0 and a 1. A pattern of the form 01100. . . is useful since it tests for the transitions 0 to 1 and 1 to 0, and the ability to hold a 0 and hold a 1.

The test vectors for this design are shown in Table 3. T1 loads zeros into the MI registers via the I1 input to all MUXs; T2 loads ones into the MI registers via the I1 input to all MUXs; T3 loads zeros via the I2 (FORCE) input to all MUXs; and T4 loads ones via the I2 (FORCE) input to all MUXs.

To load the MI registers with a more complex test pattern requires more test vectors. However, the test as proposed appears to be sufficient because it indirectly checks for hold and transition register operations; it does not test for shorts between adjacent register cells.

Since the 12P flip flops in the CSL do not form a scan chain, the bits in the test vectors must be distributed to the correct flip flops in the actual scan chain.

The testing of the MI units and associated logic consists of first scanning a test vector into the 12P flip flops of the 32 CSL units, next activating LOAD, FORCE, and FORCECTRL, and then shifting out the results from MI. Note that the 12P flip flops feed other logic and hence, later new data must be loaded to test this other logic. Overlaying these two test vector sets may be possible.

V. Add Compare Select (ACS)

Part of the logic of an ACS unit is shown in Fig. 8. The logic is driven primarily from flip flops 11P and 7P in the BFLY unit, and 13P and 14P in the METCOMP. The basic architectural structure is shown in Fig. 9. The logic in C2-ACS can be partitioned into clouds; one such cloud is shown in Fig. 10 along with the pseudo-exhaustive tests for this unit.

The testing of the MUXs 22P and 12P is straightforward, since their outputs drive scan flip flops and their inputs are either driven by primary inputs (CLOCK, WORD SYNC) or by scan flip flops.

Note that the clock input to flip flop 11P is from a MUX. During normal operation this clock is driven by the Q output of flip flop 8P. During scan mode this line is driven by CLOCK. Since the flip flops are edge triggered, a special test for this logic is necessary. One test vector is shown below.

					action
*A**	*B**	8PQ	11PQ	10P	11PQ
1	0	0	0	1	0 → 1

The scan chain is set up so that the conditions above are met. Then a normal-mode clock is issued. 8PQ will be set creating a 0 to 1 transition on the output of MUX 22P and setting flip flop 11PQ. A scan operation is then used to check the state of this flip flop. In a similar way a 0 can be loaded. No transition on the gated clock line can be produced. These conditions are summarized below.

					action	
*A**	*B**	8PQ	11PQ	10P	11PQ	8PQ
1	0	0	0	1	0 → 1	0 → 1
0	1	0	1	1	1 → 0	0 → 1
1	1	1	0	0	0 → 0	1 → 0
0	0	1	1	0	1 → 1	1 → 0

Note that these tests can be executed in the same way that other scan tests are executed, hence they are not really special.

A gate-level design of logic block C1-ACS is shown in Fig. 11(a). Also shown is a functional test set consisting of 24 vectors. The first block of vectors tests MUX 7P and establishes a sensitized path through MUX 18P, NAND gate 4P, and finally through MUX 13P. The next set of 4 vectors tests MUX 20P. The next set of 8 vectors tests the MUXs feeding NAND gate

3P. The final set of vectors tests 4P and 3P. Testing the final level of MUXs is done in a way such that all other MUXs are tested at the same time. There is some redundancy in this test set.

This circuit was processed using the USC Test Generation System (TGS). The results are shown in the Appendix. Figure A-1 shows the circuit description, which is an input to the program. Figure A-2 shows the functional test set. Figure A-3 shows the test vectors generated automatically using the PODEM algorithm. Only 16 test vectors are required to get 100 percent coverage of all single stuck-at faults. Figure A-4 shows the fault simulation results using the functional test vector set. This set also produced 100 percent fault coverage. However, 5 vectors can be deleted, reducing the test set to 19 vectors.

The discussion so far is incomplete since the ACS is not a fully scannable circuit, i.e., it contains an embedded shift register. Hence, when testing logic block C2-AC5, the results from A^* and B^* can be latched into the input to shift registers 1P and 9P. Then they can be shifted through these registers. The result from either A^* or B^*, but not both, can then be gated through MUXs 20P, 18P, and 13P into a scan flip flop SINK (see Fig. 8). This gating requires FORCE = 0 or 1, FORCE-CTRL = 1, RENORM TRIGGER = WORD SYNC = 0, K EQ 15 = 0, and ARITH CLOCK = $0 \rightarrow 1$.

Another problem exists because of these embedded shift registers. A test vector for logic block C1-ACS requires that certain values be applied to lines AP and BP. But these are outputs of the 16-bit shift registers. Hence, these values must occur at A^* and B^* 16 time periods earlier. Thus, the flip flops 11P, 7P, 13P, 14P, and the carry flip flops f20P and f30P must be set to proper values to produce the desired values of A^* and B^*. A test for C1-ACS consists of loading the scan chain with a test vector to produce the desired values of A^* and B^*, issuing 16 more clocks to drive the data through the 16-bit shift registers, and then issuing one more normal clock to load the result of the test into SINK. Then the scan chain can be read out.

To alleviate these problems, the 16-bit shift register consisting of 16 non-scan D flip flops can be modified to have the design shown in Fig. 12. Here the first and last flip flops of the shift register consist of scannable D flip flops. Now A^* and B^* are observable as part of a normal scan chain, and AP and BP are controllable as part of a normal scan chain. To test the shift register, a 0 can be scanned into 17P, and 15 normal clocks issued. The result in 20P can then be scanned out. This can then be repeated for 17P set to 1. This test is a slight modification of the normal scan test schema, in which after a scan operation, only one normal mode clock is issued.

VI. Conclusion

In this article it has been shown how the Viterbi decoder chip can be partitioned into very simple blocks of logic and test vectors generated for each such block. Most logic blocks are tested exhaustively, hence any permanent irredundant fault should be detected. It has also been indicated where normal scan design rule violations appear, and ways for overcoming these situations have been suggested.

It has not yet been determined how many scan chains should be used, which flip flops should go into which scan chains, and what the order of the flip flops in each scan chain should be. A flat design needs to be obtained so that test vectors for the entire scan chain can be determined. This will require the development of several programs, such as a test-set editor and a procedure to identify identical blocks of logic in the circuit, where in most cases each such block is a cloud. Testing parts of this circuit as a pipeline circuit is a possibility which would permit replacement of many of the scan flip flops by normal D flip flops.

Acknowledgment

The author would like to acknowledge Mr. Kuen-Jong Lee, whose efforts produced the results shown in the Appendix.

References

[1] J. Statman, G. Zimmerman, F. Pollara, and O. Collins, "A Long Constraint Length VLSI Viterbi Decoder for the DSN," *TDA Progress Report 42-95*, vol. July–September 1988, Jet Propulsion Laboratory, Pasadena, California, pp. 134–142, November 15, 1988.

[2] I. S. Hsu, "On Testing VLSI Chips for the Big Viterbi Decoder," *TDA Progress Report 42-96*, this issue.

Table 1. Logic complexity

Block	Hardware component count (inverters not counted) No. of units	Gates	MUX	FA	F/Fs	Totals Gates	F/Fs
A	144	0	0	1(5)	2	720	288
B	80	0	0	1(5)	2	400	160
C	16	0	0	0	6	0	96
D	64	0	0	0	16	0	1024
E	32	4	6(18)	0	3	704	96
F	16	2	1	0	4	48	64
G	2	0	0	0	16	0	32
H	16	30	0	0	1	480	16
						2352	1776

FA (full adder) ≡ 5 gates
MUX (multiplexer) ≡ 3 gates
EOR (exclusive or) ≡ 4 gates

Table 2. Test for logic of Figure 4

Test Vector No.	I* D 0	S* S 0	I* D 1	S* S 1	G 1	G 2	S* M 0	S* M 1	FA Inputs A = G 7	B = G 8	C* = C +	FA Outputs S	C 0
1	0	1	0	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	1	1	1	0	1	0	1	0
3	0	0	1	0	1	0	1	1	1	0	0	1	0
4	0	0	0	0	1	1	1	1	1	1	0	0	1
5	0	0	0	0	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	0	1	0	1	0	1
7	1	1	1	1	1	1	0	1	0	1	1	0	1
8	0	1	0	1	0	0	1	1	0	0	1	1	0

*Inputs

Table 3. Test for MI units

	Tests 1	2	3	4
12P	0	1	0	1
FORCE	1	0	1	0
FORCE CNTL	0	0	1	1

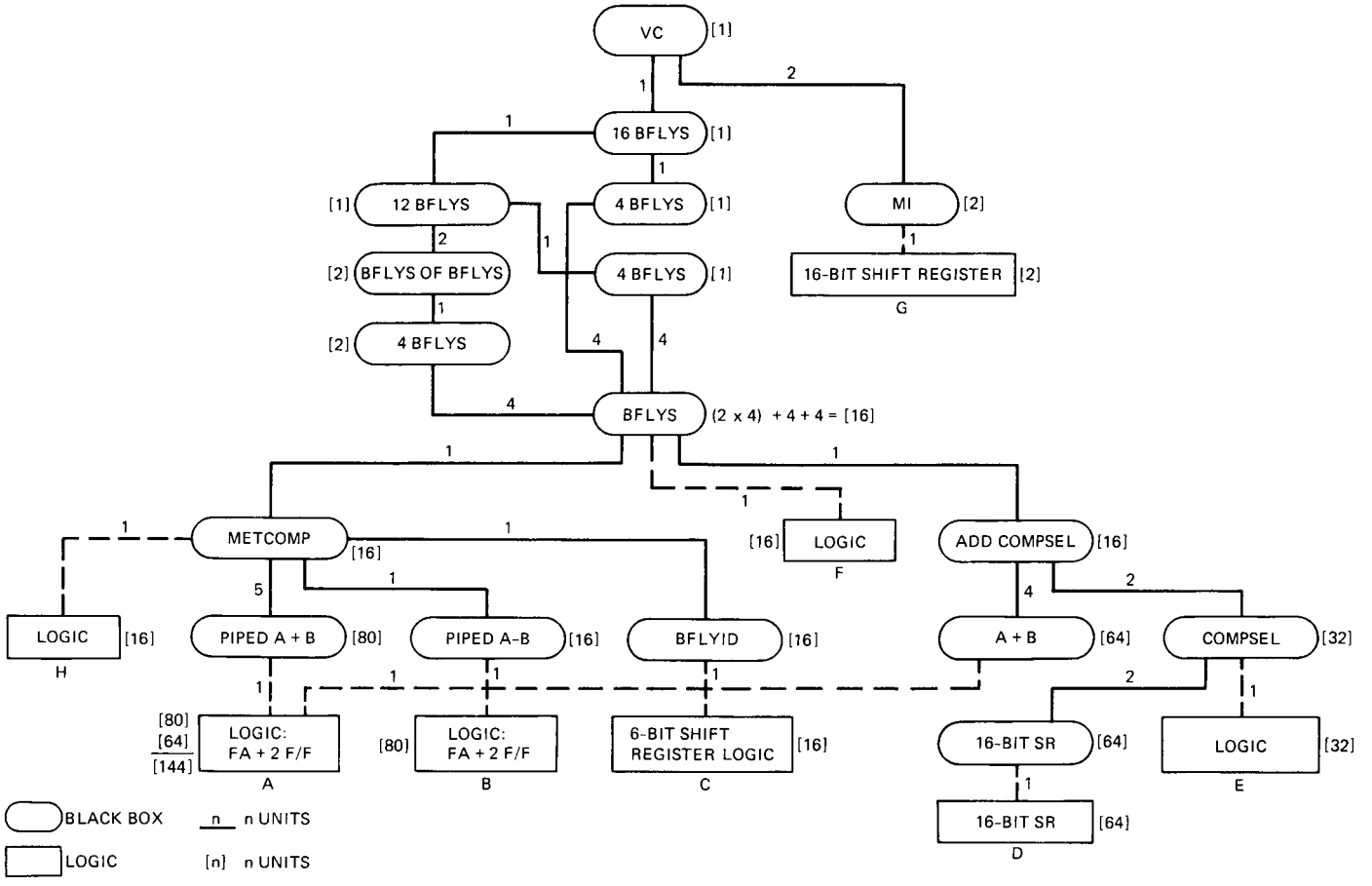


Fig. 1. Hierarchical design of Viterbi Decoder chip.

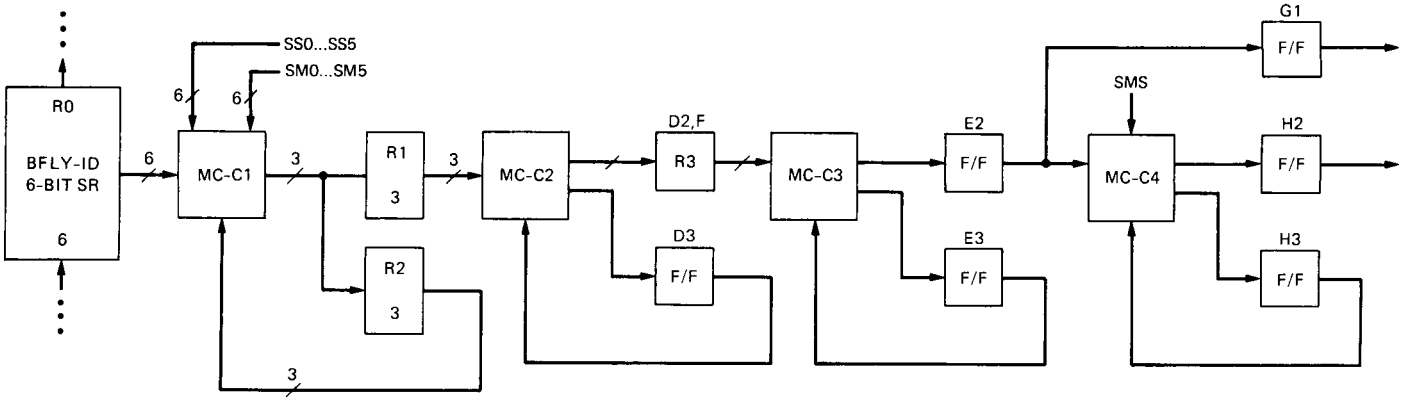


Fig. 2. General RT structure of Metric Computer.

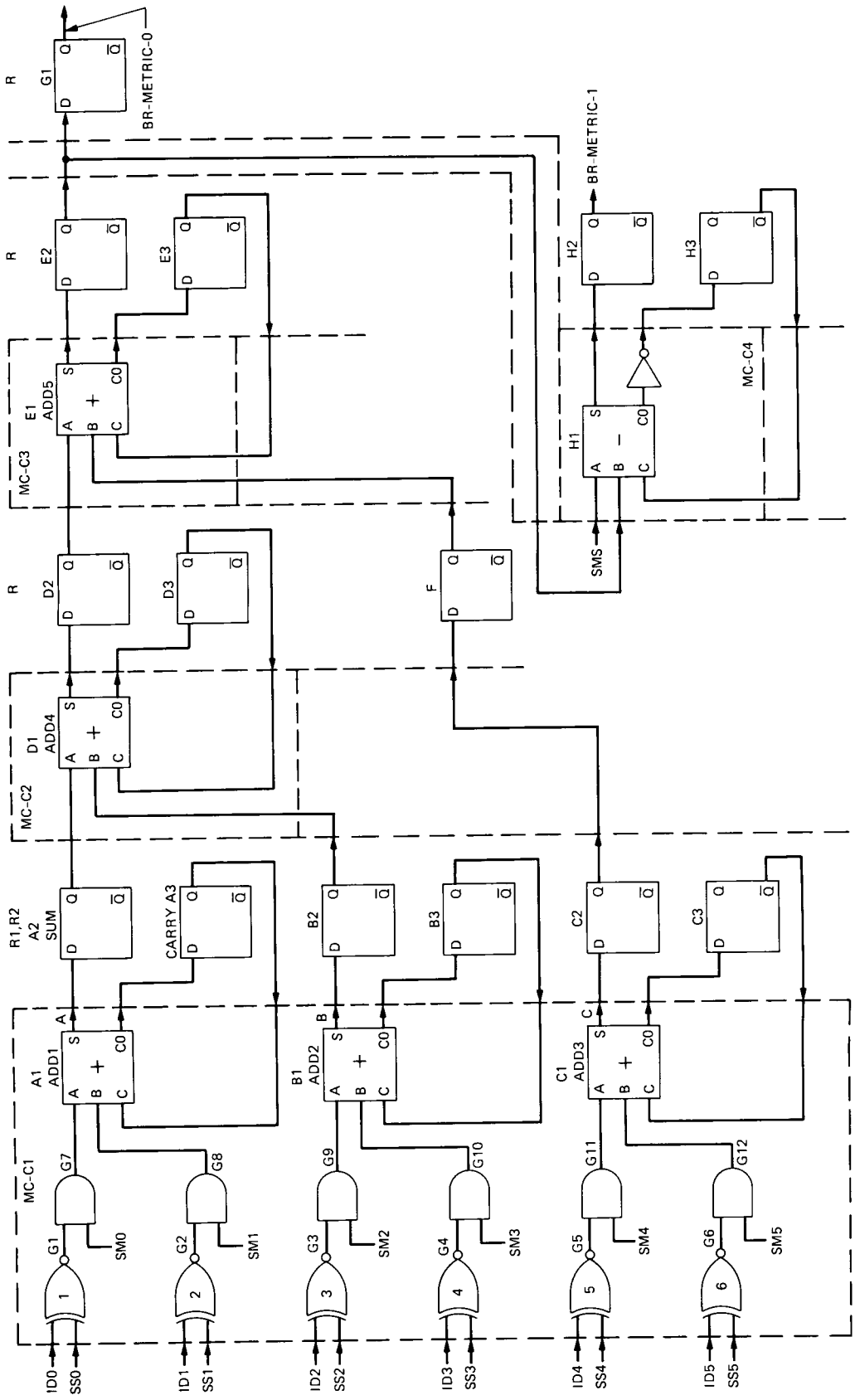


Fig. 3. Logic structure of Metric Computer.

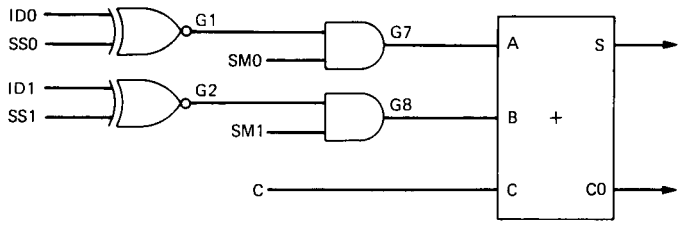


Fig. 4. One cloud in the Metric Computer.

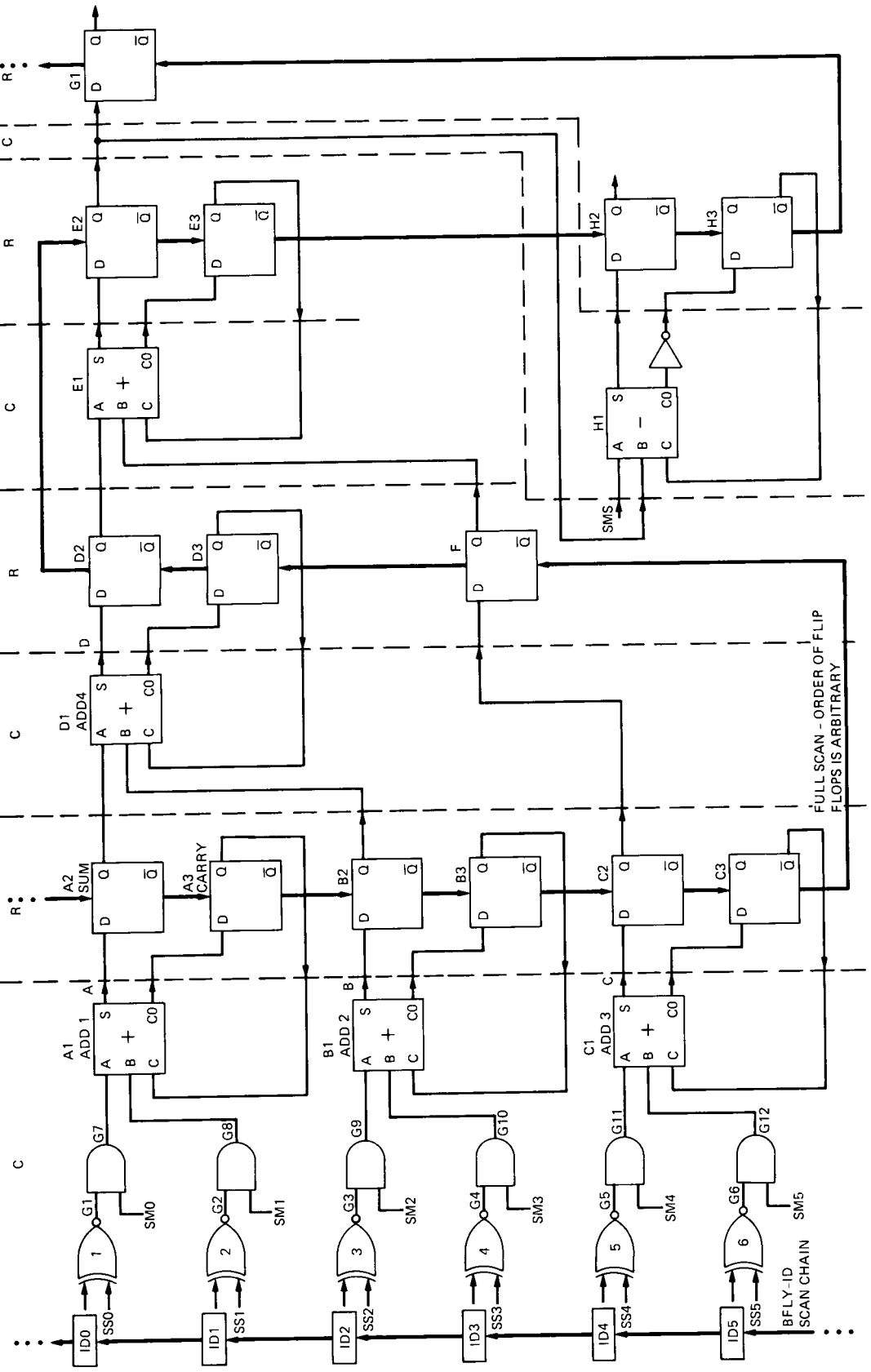


Fig. 5. Scan path in the Metric Computer.

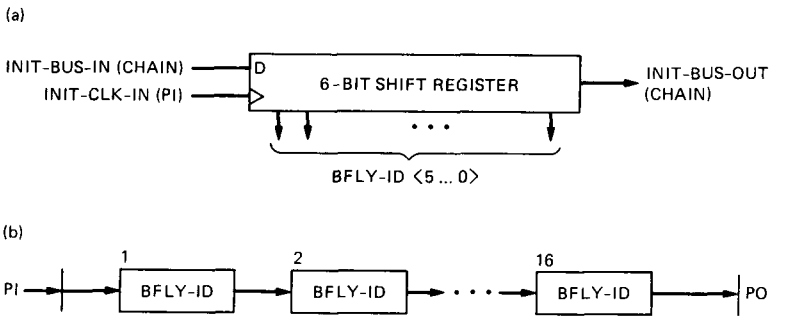


Fig. 6. BFLY-ID scan-chain architecture in the Metric Computer: (a) D flip flops forming a 6-bit shift register; and (b) 32 registers connected together to form one long shift register.

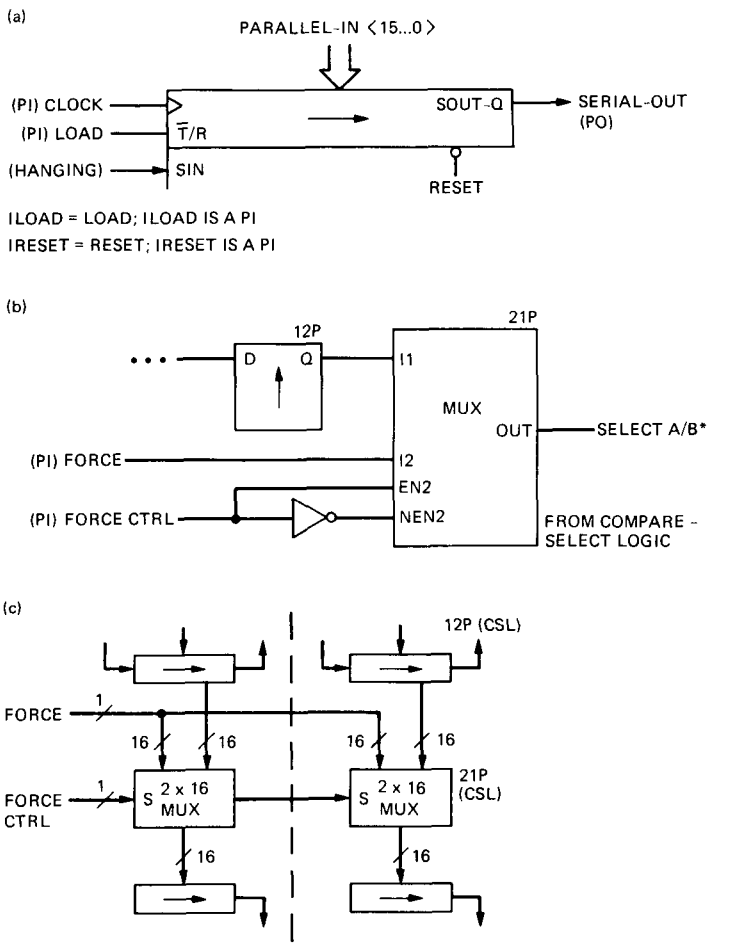


Fig. 7. Memory Interface: (a) one memory-interface unit; (b) logic driving a memory interface unit; and (c) architecture for 4 units.

FOLDOUT FRAME

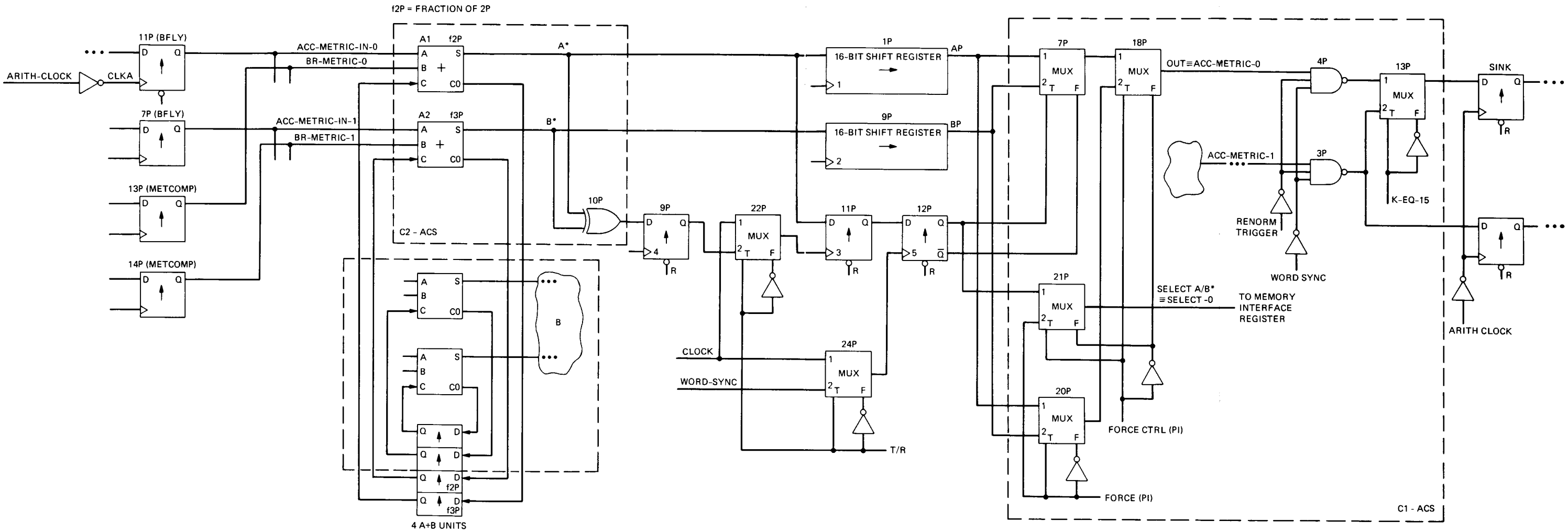


Fig. 8. Add-Compare-Select Unit block diagram.

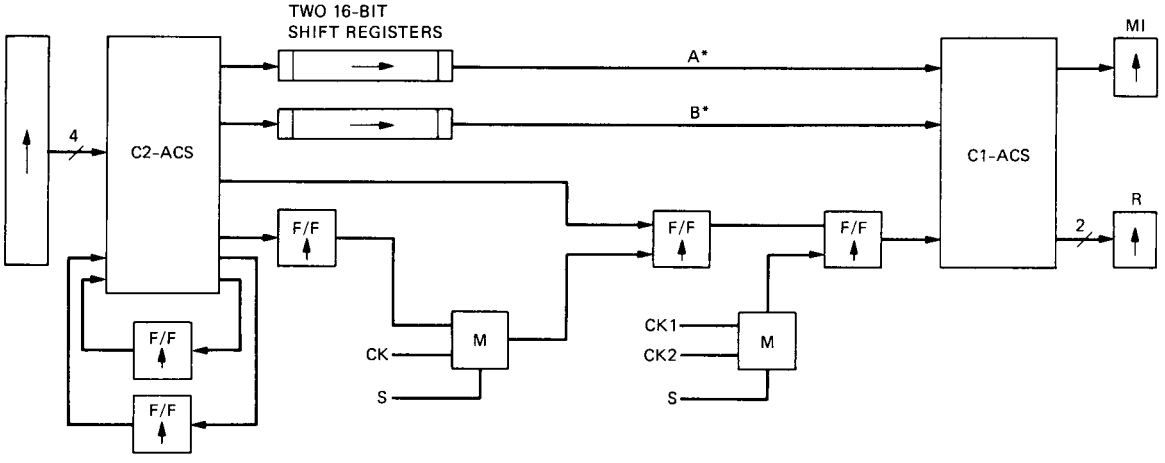
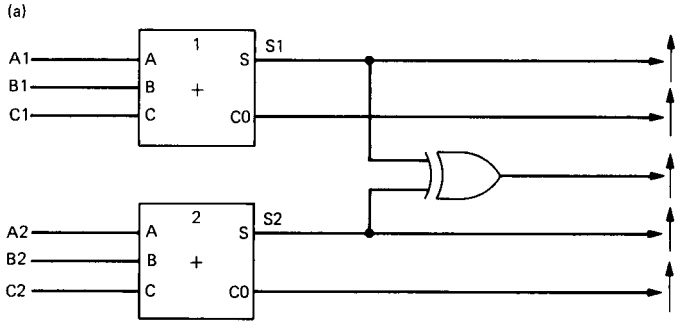


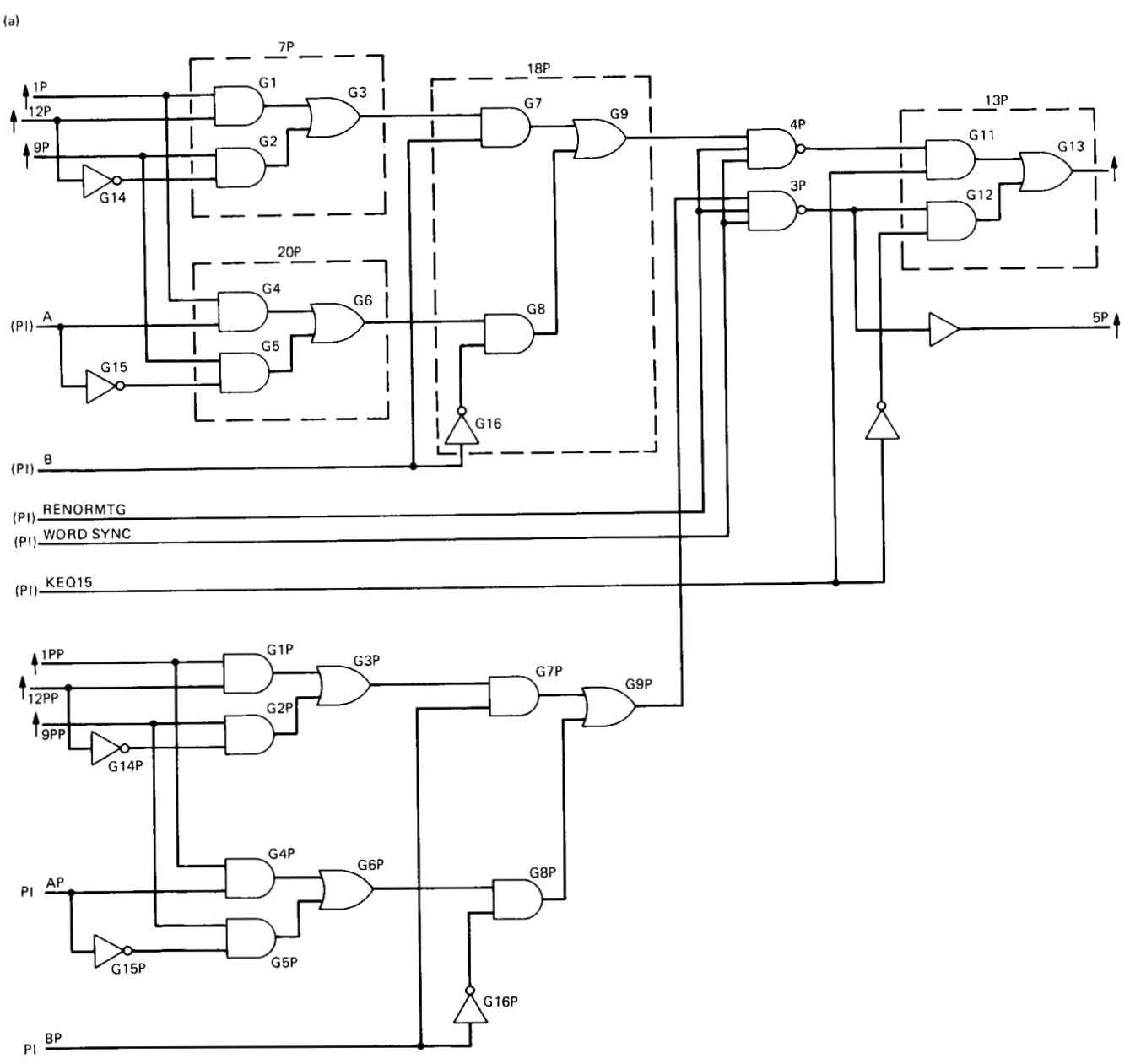
Fig. 9. Basic architectural structure of an Add-Compare-Select unit.



(b)

	A1	B1	C1	A2	B2	C2	S1	S2
T1	0	0	0	1	1	1	0	1
T2	0	0	1	0	0	0	1	0
T3	0	1	0	0	0	1	1	1
T4	0	1	1	0	1	0	0	1
T5	1	0	0	0	1	1	1	0
T6	1	0	1	1	0	0	0	1
T7	1	1	0	1	0	1	0	0
T8	1	1	1	1	1	0	1	0

Fig. 10. Logic block C2–ACS: (a) logic of a cloud and (b) test vectors. Adders 1 and 2, and the EOR gate are tested exhaustively.



(b)

RENORMTG — WORD SYNC

	1 9 12 PPPAB	KEQ15	WORD	1 9 12 PPPAB
7P	1 0 0 X 1	1 1 1		
	0 1 0 X 1	1 1 1		
	1 0 1 X 1	1 1 1		
	0 1 1 X 1	1 1 1		
20P	1 0 X 0 0	1 1 1		
	0 1 X 0 0	1 1 1		
	1 0 X 1 0	1 1 1		
	0 1 X 1 0	1 1 1		
		0 1 1		1 0 0 X 1
		0 1 1		0 1 0 X 1
		0 1 1		1 0 1 X 1
	×	0 1 1	×	0 1 1 X 1
		0 1 1		1 0 X 0 0
		0 1 1		0 1 X 0 0
		0 1 1		1 0 X 1 0
		0 1 1		0 1 X 1 0
4P	1 X 1 X 1 1 1 1	"1"		
	0 X 1 X 1 1 1 1	"0"	×	
	1 X 1 X 1 1 1 0	"1"		
	1 X 1 X 1 1 0 1	"1"		
3P		0 1 1		1 X 1 X 1
	×	0 1 1		0 X 1 X 1
		0 1 0		1 X 1 X 1
		0 0 1		1 X 1 X 1

24 FUNCTIONAL TEST VECTORS

Fig. 11. (a) Logic block C1–ACS and (b) functional test vectors.

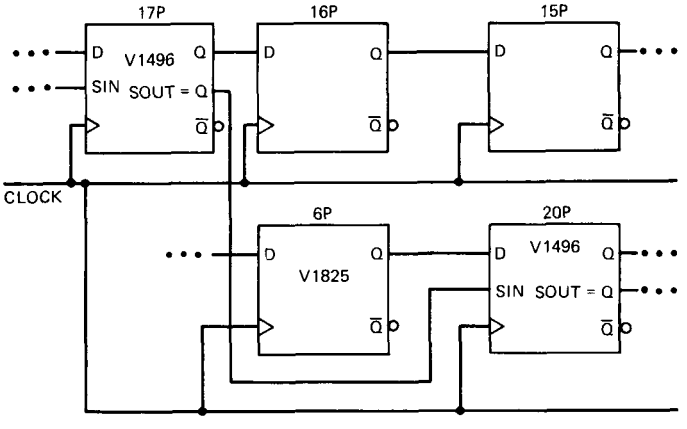


Fig. 12. Modified 16-bit shift register.

Appendix

Test Generation System Results

This Appendix contains the results of using the TGS on the combinational logic in the ACS unit. It consists of Figs. A-1 – A-4.

```
                          NO. OF    NO. OF
TYPE      NAME            FAN-OUT   FAN-IN
44
inpt      1p              2         0
inpt      9p              2         0
inpt      12p             2         0
inpt      a               2         0
inpt      b               2         0
inpt      keql5           2         0
inpt      renormtg        2         0
inpt      wordsync        2         0
inpt      1pp             2         0
inpt      9pp             2         0
inpt      12pp            2         0
inpt      ap              2         0
inpt      bp              2         0
                                              FAN-IN LIST
and       gl              1         2         1p          12p
inv       gl4             1         1         12p
and       g2              1         2         9p          gl4
or        g3              1         2         gl          g2

and       g4              1         2         1p          a
inv       gl5             1         1         a
and       g5              1         2         9p          gl5
or        g6              1         2         g4          g5

and       g7              1         2         g3          b
inv       gl6             1         1         b
and       g8              1         2         g6          gl6
or        g9              1         2         g7          g8

and       glp             1         2         1pp         12pp
inv       gl4p            1         1         12pp
and       g2p             1         2         9pp         gl4p
or        g3p             1         2         glp         g2p

and       g4p             1         2         1pp         ap
inv       gl5p            1         1         ap
and       g5p             1         2         9pp         gl5p
or        g6p             1         2         g4p         g5p

and       g7p             1         2         g3p         bp
inv       gl6p            1         1         bp
and       g8p             1         2         g6p         gl6p
or        g9p             1         2         g7p         g8p

nand      4p              1         3         g9          renormtg wordsync
nand      3p              2         3         g9p         renormtg wordsync

and       gll             1         2         4p          keql5
inv       gl7             1         1         keql5
and       gl2             1         2         3p          gl7
or        gl3             0         2         gll         gl2

buf       5p              0         1         3p
```

Fig. A-1. Circuit input description.

(a)

12345	678	1111 90123
100x1	111	xxxxx
010x1	111	xxxxx
101x1	111	xxxxx
011x1	111	xxxxx
10x00	111	xxxxx
01x00	111	xxxxx
10x10	111	xxxxx
01x10	111	xxxxx
xxxxx	011	100x1
xxxxx	011	010x1
xxxxx	011	101x1
xxxxx	011	011x1
xxxxx	011	10x00
xxxxx	011	01x00
xxxxx	011	10x10
xxxxx	011	01x10
lxlxl	111	xxxxx
0xlxl	111	xxxxx
lxlxl	110	xxxxx
lxlxl	101	xxxxx
xxxxx	011	lxlxl
xxxxx	011	0xlxl
xxxxx	010	lxlxl
xxxxx	001	lxlxl

(b)

1	1P
2	9P
3	12P
4	A
5	B
6	K EQ 15
7	RENORMTG
8	WORD SYNC
9	1PP
10	9PP
11	12PP
12	AP
13	BP

Fig. A-2. (a) Functional test vectors and (b) corresponding column headings.

```
Please enter circuit file name: cfunc


                MAIN MENU

            0. Exit
            1. Fault-collapsing
            2. Test-generation
            3. Fault-simulation
            4. Logic-simulation
            5. Integrated System

          Please enter your choice: 5

 Would you like to use the following default values?

    Exiting Condition : fault coverage = 100%
    In-order fault selection
    Test Generation Method : PODEM

   Please enter: [y/n] y

For the following file names, enter
 <RETURN> to use default file name as shown in parentheses,
    "/"   to suppress file generation, or
  enter the desired name.

Please enter file name for fault classes.
 (default name: cfunc.cls) :
Please enter file name for resulting test vectors.
 (default name: cfunc.tst) :
Please enter file name for fault list.
 (default name: cfunc.flt) :
Please enter file name for complete output result.
 (default name: cfunc.res) :
Please enter file name for execution time.
 (default name: cfunc.tim) :

If there is any "x" (don't care) in the test vector,
 what value should it be assigned?
    1. always assign "1"
    2. always assign "0"
    3. randomly assign "1" or "0"
 Enter your selection: 3
 Enter the probability of assigning "1": 0.6

 Please choose one option for fault collapsing:
   1  Equivalence merging only (for circuits with feedback).
   2  Equivalence as well as dominance merging.
 Enter option: 2

**** Fault Collapsing OK! ****

 Number of faults = 144

*** Now enter the main loop ... ***

Number of detected faults = 0
Current fault coverage is 0.00%

**************************************************
**************************************************
 First selection iteration..
------------------------------------------
Vector[1] is
11101 11111 111
Number of detected faults = 29
Current fault coverage is 20.14%
------------------------------------------
Vector[2] is
01111 11101 111
Number of detected faults = 58
Current fault coverage is 40.28%
------------------------------------------
Vector[3] is
10011 11100 110
Number of detected faults = 69
Current fault coverage is 47.92%
------------------------------------------
Vector[4] is
11011 11110 100
Number of detected faults = 83
Current fault coverage is 57.64%
------------------------------------------
Vector[5] is
10010 11100 101
Number of detected faults = 90
Current fault coverage is 62.50%
------------------------------------------
Vector[6] is
01110 11110 101
Number of detected faults = 97
Current fault coverage is 67.36%
------------------------------------------
Vector[7] is
10100 11110 110
Number of detected faults = 109
Current fault coverage is 75.69%
------------------------------------------
Vector[8] is
11000 11111 101
Number of detected faults = 113
Current fault coverage is 78.47%
------------------------------------------
Vector[9] is
10110 01010 011
Number of detected faults = 117
Current fault coverage is 81.25%
------------------------------------------
Vector[10] is
10001 11110 011
Number of detected faults = 123
Current fault coverage is 85.42%
------------------------------------------
Vector[11] is
10110 01111 011
Number of detected faults = 129
Current fault coverage is 89.58%
------------------------------------------
Vector[12] is
11101 01101 010
Number of detected faults = 132
Current fault coverage is 91.67%
------------------------------------------
Vector[13] is
11010 11111 000
Number of detected faults = 136
Current fault coverage is 94.44%
------------------------------------------
Vector[14] is
11001 10110 101
Number of detected faults = 139
Current fault coverage is 96.53%
------------------------------------------
Vector[15] is
01011 11011 100
Number of detected faults = 142
Current fault coverage is 98.61%
------------------------------------------
Vector[16] is
00011 01101 011
Number of detected faults = 144
Current fault coverage is 100.00%
**************************************************
**************************************************
 Total number of faults is 144.

 144 faults have been detected by 16 test vectors.
 The fault coverage is 100.0000 %.

**************************************************
```

Fig. A-3. Results of automatic test-vector generation.

```
Please enter circuit file name: cfunc


                    MAIN MENU

                0. Exit
                1. Fault-collapsing
                2. Test-generation
                3. Fault-simulation
                4. Logic-simulation
                5. Integrated System

Please enter your choice: 5

Would you like to use the following default values?

    Exiting Condition : fault coverage = 100%
    In-order fault selection
    Test Generation Method : PODEM

Please enter: [y/n] n

Which Test Generation method should be used?
    1. PODEM
    2. Random Test Generation
    3. Test Vectors in a file
Enter your selection: 3

Which exiting condition do you wish to use ?
    1. fault coverage
    2. number of test vectors
    3. CPU time (not available)
Enter your selection: 1

Please enter the percentage fault coverage desired: 100

                FUNCTIONAL TEST VECTORS

Please enter input file name for test vectors.
 <RETURN> --- use default name: cfunc.int
    Enter:
For the following file names, enter
 <RETURN> to use default file name as shown in parentheses,
    "/"   to suppress file generation, or
  enter the desired name.

Please enter file name for fault classes.
 (default name: cfunc.cls) : cfuncl.cls
Please enter file name for resulting test vectors.
 (default name: cfunc.tst) : cfuncl.tst
Please enter file name for fault list.
 (default name: cfunc.flt) : cfuncl.flt
Please enter file name for complete output result.
 (default name: cfunc.res) : cfuncl.res
Please enter file name for execution time.
 (default name: cfunc.tim) : cfuncl.tim

If there is any "x" (don't care) in the input vector,
what value should it be assigned?
    1. always assign "1"
    2. always assign "0"
    3. randomly assign "1" or "0"
Enter your selection: 3
Enter the probability of assigning "1": 0.6

Please choose one option for fault collapsing:
    1  Equivalence merging only (for circuits with feedback).
    2  Equivalence as well as dominance merging.
Enter option: 2

**** Fault Collapsing OK! ****

Number of faults = 144

*** Now enter the main loop ... ***

Number of detected faults = 0
Current fault coverage is 0.00%
-----------------------------------------------
Vector[1] is
10011 11101 111
Number of detected faults = 31
Current fault coverage is 21.53%
-----------------------------------------------
Vector[2] is
01011 11111 011
Number of detected faults = 62
Current fault coverage is 43.06%
-----------------------------------------------
Vector[3] is
10111 11111 001
Number of detected faults = 66
Current fault coverage is 45.83%
-----------------------------------------------
Vector[4] is
01111 11101 110
Number of detected faults = 78
Current fault coverage is 54.17%
-----------------------------------------------
Vector[5] is
10100 11100 000
Number of detected faults = 88
Current fault coverage is 61.11%
-----------------------------------------------
Vector[6] is
01000 11110 111
Number of detected faults = 100
Current fault coverage is 69.44%
-----------------------------------------------
Vector[7] is
10110 11101 011
Number of detected faults = 103
Current fault coverage is 71.53%
-----------------------------------------------
Vector[8] is
01110 11101 101
Number of detected faults = 110
Current fault coverage is 76.39%
-----------------------------------------------
Vector[9] is
01110 01110 011
Number of detected faults = 117
Current fault coverage is 81.25%
-----------------------------------------------
Vector[10] is
10110 01101 011
Number of detected faults = 118
Current fault coverage is 81.94%
-----------------------------------------------
Vector[11] is
10011 01110 111
Number of detected faults = 120
Current fault coverage is 83.33%
-----------------------------------------------
*Vector[12] is
 00011 01101 111
 Number of detected faults = 120
 Current fault coverage is 83.33%
-----------------------------------------------
 Vector[13] is
 10110 01110 000
 Number of detected faults = 122
 Current fault coverage is 84.72%
-----------------------------------------------
 Vector[14] is
 11111 01101 000
 Number of detected faults = 130
 Current fault coverage is 90.28%
-----------------------------------------------
 Vector[15] is
 10011 01110 010
 Number of detected faults = 133
 Current fault coverage is 92.36%
-----------------------------------------------
*Vector[16] is
 10110 01101 110
 Number of detected faults = 133
 Current fault coverage is 92.36%
-----------------------------------------------
 Vector[17] is
 10111 11101 010
 Number of detected faults = 138
 Current fault coverage is 95.83%
-----------------------------------------------
*Vector[18] is
 01111 11111 000
 Number of detected faults = 138
 Current fault coverage is 95.83%
-----------------------------------------------
 Vector[19] is
 11101 11010 100
 Number of detected faults = 140
 Current fault coverage is 97.22%
-----------------------------------------------
 Vector[20] is
 11111 10101 010
 Number of detected faults = 142
 Current fault coverage is 98.61%
-----------------------------------------------
*Vector[21] is
 01101 01110 111
 Number of detected faults = 142
 Current fault coverage is 98.61%
-----------------------------------------------
*Vector[22] is
 11101 01101 101
 Number of detected faults = 142
 Current fault coverage is 98.61%
-----------------------------------------------
 Vector[23] is
 10000 01010 111
 Number of detected faults = 143
 Current fault coverage is 99.31%
-----------------------------------------------
 Vector[24] is
 11100 00110 101
 Number of detected faults = 144
 Current fault coverage is 100.00%
 ************************************************
 ************************************************
  Total number of faults is 144.

  144 faults have been detected by 24 test vectors.
  The fault coverage is 100.0000 %.

 ************************************************
 *These vectors can be deleted.
```

Fig. A-4. Results of fault simulation for functional test vectors.